

LABIA BLOCKS: BLOCK-BASED PROGRAMMING, ARTIFICIAL INTELLIGENCE AND GAMIFICATION IN INTRODUCTORY WEB GAME EDUCATION

 <https://doi.org/10.63330/aurumpub.057-026>

Tiago Saidelles¹ and Miguel Augusto Bauermann Brasil²

Abstract

This study presents and analyzes LabIA Blocks, an educational platform developed to support introductory programming and digital game creation by integrating block-based programming, HTML, CSS, JavaScript, gamification, and artificial intelligence. This is an applied, descriptive, and technological development study whose objective is to describe the construction of the tool, its features, and its potential pedagogical applications for teachers and students. The methodology comprised a functional and documentary analysis of the final version of the system, an examination of the block catalog and the teacher and student areas, and the association of the platform's features with the principles of active methodologies, computational thinking, and project-based learning. The analysis revealed an environment that combines a visual editor with code generation, a contextual AI Tutor, planning in Portugol, progressive challenges, ready-made games, original game creation, and teaching resources for organizing classes, registering students, publishing activities, and monitoring submissions. The expected outcomes of its pedagogical use include fostering a gradual understanding of programming logic, experimentation, debugging, autonomy, and student agency by connecting introductory web development concepts with practical game-creation scenarios. The study concludes that LabIA Blocks has potential as an integrated teaching resource for active and gamified programming learning experiences.

¹ Master's Degree in Professional and Technological Education
Universidade Federal de Santa Maria (UFSM)
Email: tiago-saidelles@redes.ufsm.br
Lattes: <http://lattes.cnpq.br/786602013044509>

² PhD in Rural Extension
Universidade Federal de Santa Maria (UFSM)
Email: miguelbrasil@ctism.ufsm.br
Lattes: <http://lattes.cnpq.br/9617974701932533>

Keywords: Active methodologies, Artificial intelligence in education, Block-based programming, Game development, Gamification.

INTRODUCTION

Learning to program during the initial stages of contact with computing involves a twofold challenge: understanding abstract concepts such as variables, events, conditions, repetition, and problem decomposition while simultaneously mastering the syntax of a language. This combination may increase the cognitive load placed on novice students and shift their attention away from the logic of the problem toward the formal details of writing code. In this context, visual block-based programming environments offer an alternative that makes program structure more visible and manipulable without eliminating the need to understand relationships involving cause, sequence, condition, and state. The literature on computational thinking specifically emphasizes the importance of problem-solving, modeling, and solution-building practices as part of computing education (Grover; Pea, 2013, pp. 38–43).

Visual environments designed for creating stories, animations, and games have helped establish block-based programming as an introductory strategy. Scratch, for example, was designed to encourage the creation, experimentation, and remixing of projects, bringing programming closer to meaningful forms of original creation (Resnick et al., 2009, pp. 60–67). In a comparative study involving high school students, Weintrop and Wilensky (2018, pp. 1–25) observed learning gains in both text-based and block-based environments, with the block-based environment yielding favorable results for certain indicators of learning and future interest in computing. Such evidence does not support the claim that blocks are universally superior, but it does support their use as an initial form of mediation and as a bridge to text-based languages.

In web development, introductory learning can be enriched when students understand the distinct yet complementary roles of HTML, CSS, and JavaScript. HTML organizes the structure and elements of a page; CSS defines its appearance, positioning, and responsiveness; and JavaScript introduces behaviors,

events, variables, conditions, and game rules. This separation promotes a modular understanding of the system and makes it possible to relate each visual block to an equivalent segment of textual code. When creating a game, students move beyond manipulating isolated commands and begin integrating objectives, characters, rules, states, scoring, collisions, victory, and defeat into an executable artifact.

Game creation also provides opportunities to integrate programming with active methodologies. Rather than merely receiving expository instruction, students can formulate a problem, design a solution, test hypotheses, debug errors, revise their projects, and share a product. The literature on active learning shows that strategies in which students participate cognitively in solving tasks may produce better outcomes than exclusively expository approaches in science, technology, engineering, and mathematics contexts (Freeman et al., 2014, pp. 8410–8415). At the same time, game elements applied in nongame contexts, such as challenges, progression, feedback, achievements, and goals, correspond to the concept of gamification proposed by Deterding et al. (2011, pp. 9–15).

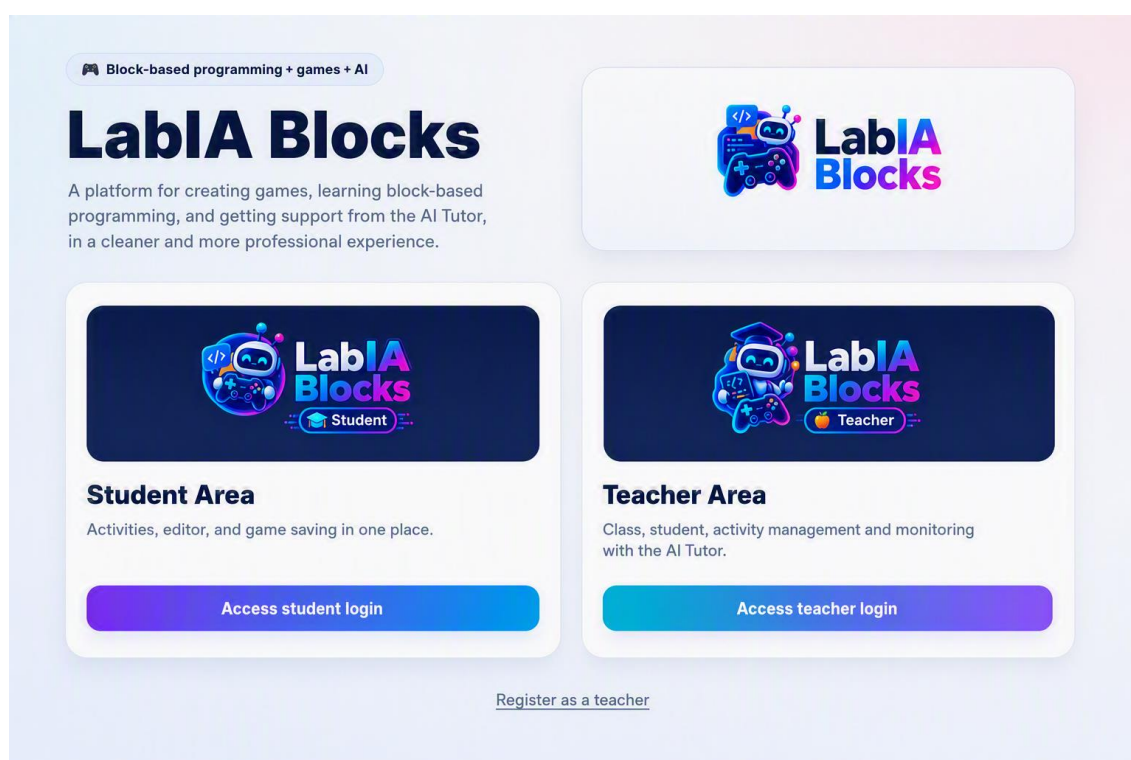
The recent incorporation of generative artificial intelligence expands the possibilities for individualized support during programming activities, but it also introduces risks related to dependency, inaccurate responses, and the replacement of students' reasoning. In a review of generative AI in computing education, Humble (2024) identifies opportunities related to instructional support, personalization, motivation, and knowledge expansion, alongside risks involving inaccuracy, overreliance, and loss of skills. UNESCO (2024) recommends that the educational adoption of AI be guided by human, ethical, and pedagogical criteria, with particular attention to privacy, system suitability, and the development of learner autonomy.

LabIA Blocks is situated within this context. The platform was developed as an integrated educational environment for creating web games through blocks, combining a visual editor, textual code, game execution, planning resources, structured activities, and an AI Tutor. The research problem guiding this study may be formulated as follows: How can a platform that integrates block-based programming, game development, gamification, and AI support organize introductory learning experiences involving

HTML, CSS, and JavaScript? The general objective is to describe the construction process and features of LabIA Blocks and analyze its pedagogical potential for introductory programming and game development education. The specific objectives are to characterize its architecture, identify resources intended for teachers and students, map the types of activities supported by the tool, and discuss the qualities, potential benefits, and limitations of its educational use.

Figure 1

LabIA Blocks home screen providing access to the student and teacher areas



Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026); Interface text translated and image adapted.

METHODOLOGY

NATURE AND RESEARCH DESIGN

This study is characterized as applied, descriptive, and technological development research. The object of analysis is LabIA Blocks itself, considered as a digital educational artifact. At this stage, no classroom experiment, questionnaire administration, or data collection involving human participants was conducted. The results presented therefore refer to the functional and pedagogical analysis of the system

rather than to evidence of learning effectiveness within a specific population. An empirical evaluation involving students and teachers is recommended as a subsequent stage and, when conducted, must comply with the applicable ethical procedures.

DEVELOPMENT PROCESS AND ARCHITECTURE

The analyzed version was structured as a web application using PHP 8 or later, MySQL/MariaDB, Bootstrap, CSS, and JavaScript. The system is organized around two primary user profiles: teacher and student. Teachers administer classes, students, activities, submissions, and AI Tutor settings. Students access published activities, open the creation environment, save projects, and interact with the AI Tutor within the context of the work underway. The game editor is embedded in the student area and operates locally for assembling and executing blocks, while services such as authentication, projects, activities, and submissions use the database.

The programming library is organized through a block catalog used by both the editor and the AI Tutor's validation mechanism. In the version examined, 108 blocks were identified: 16 HTML blocks, 15 CSS blocks, and 77 JavaScript blocks. The HTML blocks encompass structure, text, buttons, fields, images, scoreboards, messages, canvas elements, lists, and footers. The CSS blocks cover pages, Flexbox, Grid, panels, text, buttons, fields, scoreboards, canvas elements, positioning, shadows, effects, animations, media queries, and custom CSS. The JavaScript set includes events, movement, appearance, sound, control, variables, lists, interface updates, scoring, lives, victory, and defeat, in addition to specific mechanics for platform games and castle settings.

The AI Tutor runs on the server. The API key configured by the teacher is not exposed in the student's HTML and is stored in encrypted form in the teacher's settings. The Tutor's logic uses the platform's actual block catalog to avoid inventing block names. It was structured to interpret the student's request; associate entities, events, conditions, actions, and variables; verify causal order; and produce a sequence of blocks indicating the fields to be completed and how the blocks should be connected. When a

mechanic cannot be verified using the available blocks, the system is designed to indicate this limitation rather than present an assembly as correct.

FUNCTIONAL ANALYSIS PROCEDURES

The analysis was conducted using the update package made available on August 13, 2026. The configuration files, block catalog, teacher and student area pages, visual editor, activity, project, submission, and AI Tutor modules, as well as the Portugol planning routines, challenges, and ready-made games, were examined. The principal interfaces were subsequently run locally to capture screen images and verify the options visible to users. The features were classified according to four dimensions: introduction to programming, game creation, pedagogical management, and support/gamification. The pedagogical discussion was developed by establishing connections between the features actually present in the system and the literature on computational thinking, block-based programming, active learning, gamification, and generative AI in education. Because this was not a user evaluation, terms such as benefit, quality, and potential are used to denote pedagogical possibilities supported by the tool’s design rather than causal effects that have already been demonstrated.

Table 1
Technical Components and Educational Functions of LabIA Blocks

Component	Function within the Tool	Potential Pedagogical Contribution
Block editor	Visual assembly of HTML, CSS, and JavaScript	Reduce initial syntactic barriers and make logic manipulable.
Generated code	Display the textual code corresponding to the blocks	Support the transition between visual representation and textual language.
Game preview	Run and test the project within the environment	Promote a build-test-debug-revise cycle.
AI Tutor	Answer questions, explain concepts, provide hints, and analyze logic	Offer contextual support and feedback during problem-solving.
Portugol	Plan algorithms and relate them to blocks	Encourage decomposition and logical organization before implementation.
Challenges	Provide a progressive sequence with verifiable objectives	Structure progression, feedback, and learning goals.
Ready-made games	Provide complete models that can be loaded and modified	Enable analysis, remixing, and learning from examples.
Teacher/student portal	Manage activities, projects, submissions, and feedback	Integrate student authorship with teacher monitoring.

Source: Prepared by the authors (2026).

RESULTS AND DISCUSSION

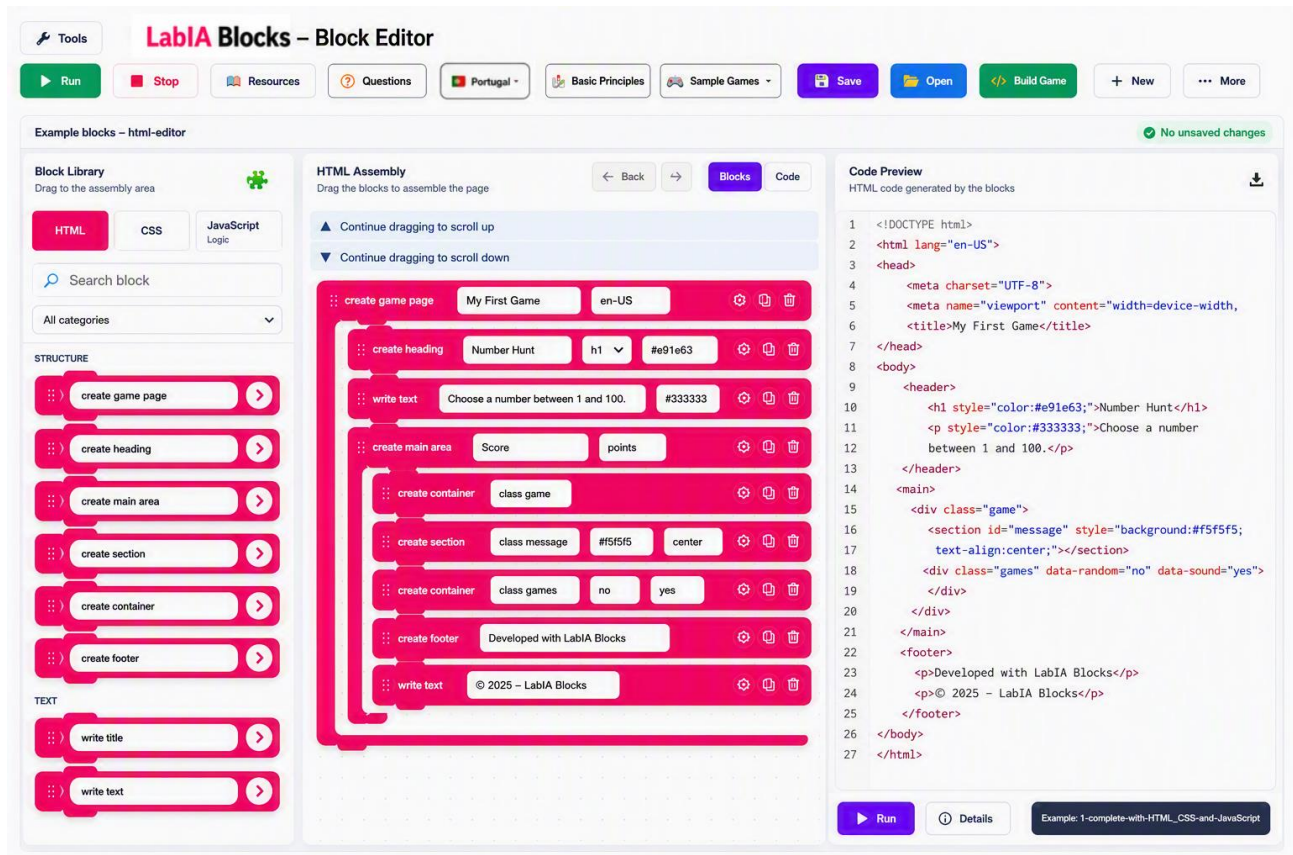
CREATION ENVIRONMENT AND ORGANIZATION OF LANGUAGES

The creation area consists of three complementary spaces: the block library, the assembly area, and the game preview. Students select HTML, CSS, or JavaScript and work with blocks differentiated by language and category. This visual separation reinforces the function of each layer of a web game: structure in HTML, appearance in CSS, and behavior in JavaScript. The ability to switch between the block view and textual code creates a particularly relevant pedagogical bridge for introductory classes because students can observe how a visually implemented decision is translated into syntax.

The blocks contain editable fields for values such as identifiers, classes, text, keys, coordinates, sizes, colors, variable names, operators, and quantities. This characteristic prevents block-based programming from being reduced to the mechanical connection of pieces. To obtain the expected behavior, students must understand the meaning of the parameters. In a movement activity, for example, a keyboard event must be associated with a coherent change in the X or Y coordinate. In a collection activity, character names, variables, and collision conditions must be interconnected. The system also provides undo and redo functions, block search, category filters, execution and restart functions, generation of the game as HTML, and the saving and opening of projects in JSON format.

Figure 2

Block-based programming and visualization of the corresponding HTML, CSS, and JavaScript code



Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026); Interface text translated and image adapted.

The simultaneous presence of blocks, code, and visual output may support an experimental learning approach. Students modify a value, run the game again, and observe the consequence. This cycle resembles the construction and debugging practices discussed in relation to creative programming environments (Resnick et al., 2009, pp. 60–67) and prevents code from being presented merely as an abstract sequence of commands. The tool does not eliminate the complexity of programming; rather, it reorganizes that complexity into more visible and manipulable stages.

THE AI TUTOR AS CONTEXTUAL PEDAGOGICAL SUPPORT

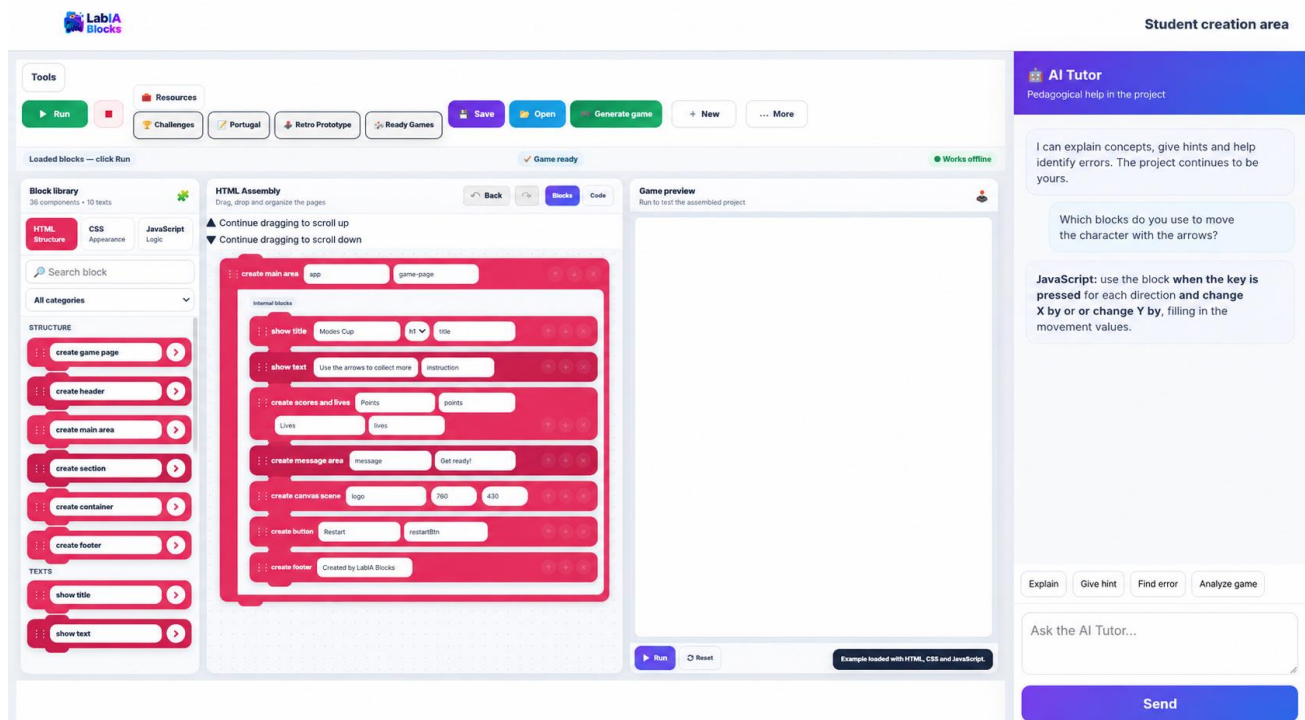
The AI Tutor is one of the platform's distinguishing features. It is integrated into the creation area and offers quick actions for explaining the project, providing hints, identifying possible errors, and

analyzing gameplay. Students may also ask open-ended questions. The pedagogical objective stated in the interface is to help students understand concepts and identify problems without depriving them of authorship. This design is consistent with a view of AI as a mediator rather than a substitute for learners' intellectual activity.

For questions concerning game construction, the Tutor uses the platform's block catalog. The expected response is not limited to a general recommendation to use HTML, CSS, or JavaScript. It must present the actual names of the blocks, the order in which they should be used, the fields that must be completed, suggested values, and the relationships among block connections. The validation mechanism attempts to relate phrases from the request to a logical contract by identifying entities, triggers, conditions, actions, states, victory, and defeat. This approach is important in school settings because it reduces the discrepancy between the AI's textual guidance and what students can actually find in the interface.

Figure 3

Example of the creation area with the AI Tutor integrated into the student's project



Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

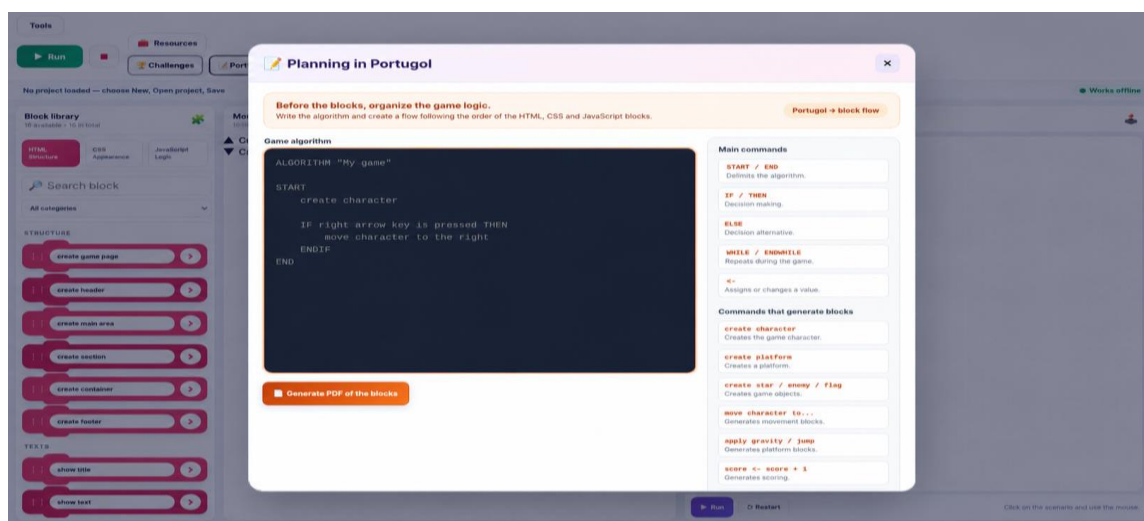
Recent literature recommends caution in this integration. Humble (2024) identifies generative AI as a potential teaching and learning assistant and as a resource for personalization and motivation, while also drawing attention to inaccuracy, overreliance, difficulty with complex tasks, and loss of skills. UNESCO (2024) emphasizes pedagogical validation, data protection, and human-centeredness. In LabIA Blocks, two design decisions address these concerns: the API key remains on the server, and teachers can control the availability and level of AI assistance for each activity. Nevertheless, the quality of the responses must continue to be monitored, particularly because generative systems cannot guarantee absolute accuracy in every situation.

PLANNING IN PORTUGOL AND TRANSITIONING TO BLOCKS

The Portugol module introduces a stage that precedes game assembly. Students can write an algorithm in a structured language, validate commands, view relationships with the blocks, and generate an outline. The resource includes examples of commands such as start/end, create character, create platform, create star, apply gravity, move character, and win the game. It also allows students to generate a PDF containing the related blocks and convert recognized commands into block trees within the project.

Figure 4

LabIA Blocks Portugol planning module

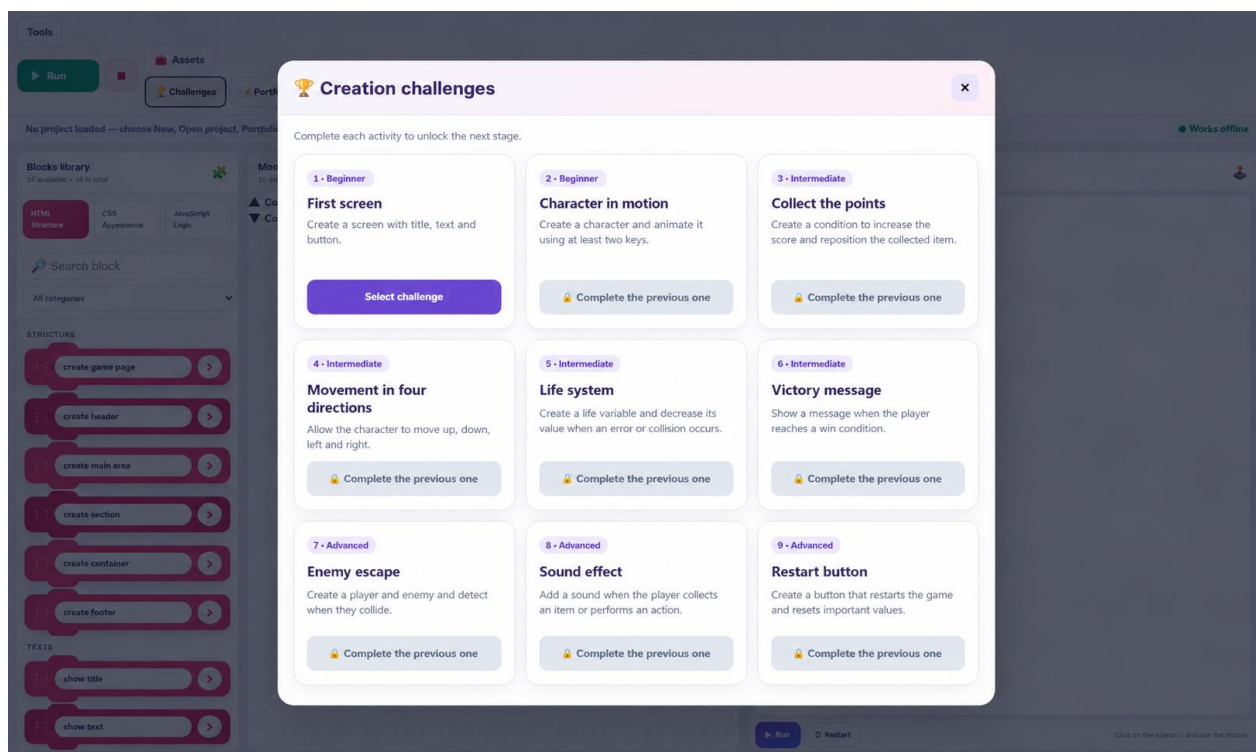


Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

From a pedagogical perspective, Portugol can function as an intermediate layer between the problem narrative and visual implementation. Instead of beginning directly in the interface, students are invited to set out a logical sequence explicitly. This practice encourages problem decomposition and the anticipation of conditions before implementation. When combined with the ability to alternate between blocks and code, the tool offers three representations of the same reasoning: algorithmic description, visual assembly, and textual code. Teachers can use this multiplicity to compare different ways of representing a solution and discuss their equivalences and limitations.

CHALLENGES, PROGRESSION, AND GAMIFICATION

LabIA Blocks provides a sequence of gradually progressing challenges. In the analyzed version, the identified activities included First Screen, Moving Character, Point Hunt, Movement in Four Directions, Life System, Victory Message, Escape the Enemy, Sound Effect, Restart Button, Complete Game, and Star Adventure. Each challenge contains verifiable objectives, and access to the next challenge may depend on completing the preceding one. The platform records completion and execution, provides feedback on objectives that have been met or remain pending, and offers achievements associated with use of the environment.

Figure 5*Progressive challenge screen for game creation*

Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

These elements constitute an application of gamification because they incorporate progression, goals, feedback, and achievements into an educational activity whose purpose is not simply to play but to learn programming. According to the definition provided by Deterding et al. (2011, pp. 9–15), gamification involves the use of game design elements in nongame contexts. Its pedagogical value, however, depends on how these elements are integrated with the learning objective. In LabIA Blocks, a challenge is considered complete based on the presence and execution of programming elements. This makes it possible to link the reward to the actual construction of a solution rather than merely to superficial participation.

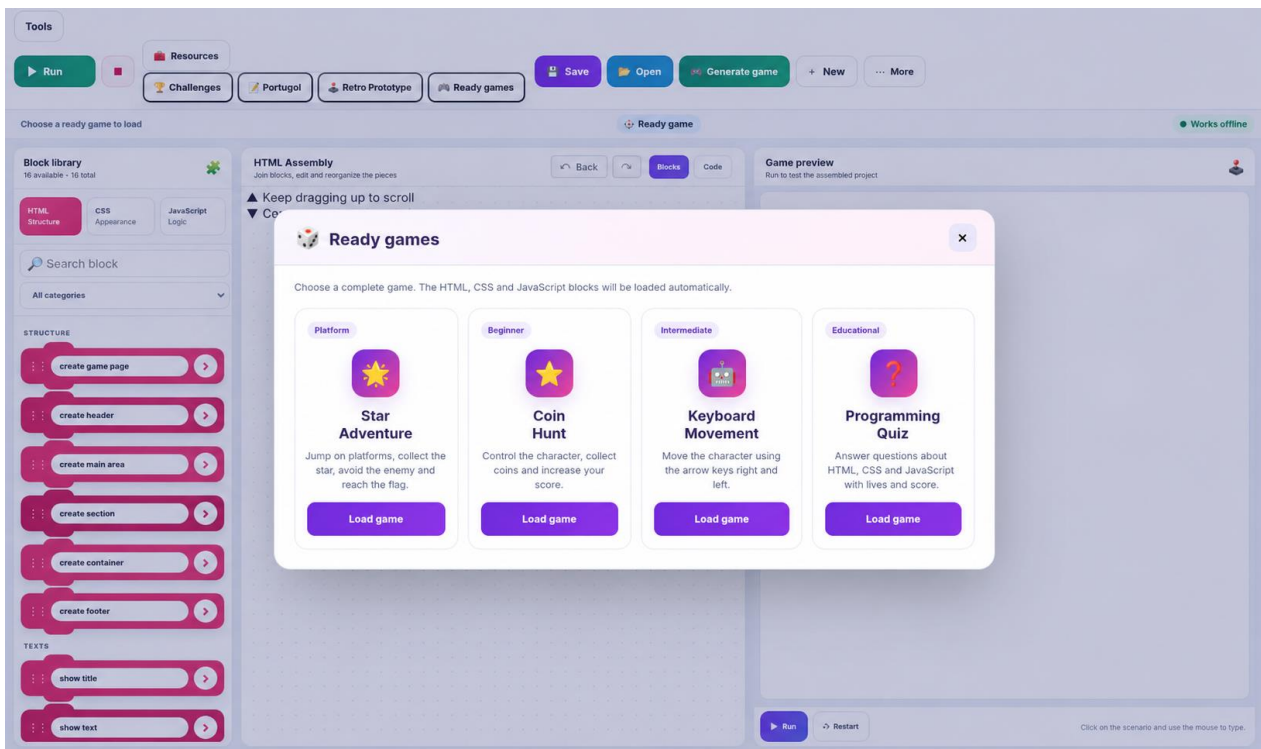
READY-MADE GAMES, REMIXING, AND ORIGINAL CREATION

The Ready-Made Games feature provides complete models that can be loaded automatically. The analyzed version includes Star Adventure, Coin Hunt, Keyboard Movement, and Programming Quiz. The

models encompass different levels and mechanics, allowing students to begin with a functional implementation and then examine the blocks, change parameters, remove components, and create variations. This approach can be used for code-reading activities, remixing, comparing strategies, or extension challenges.

Figure 6

Library of ready-made games available for analysis and remixing



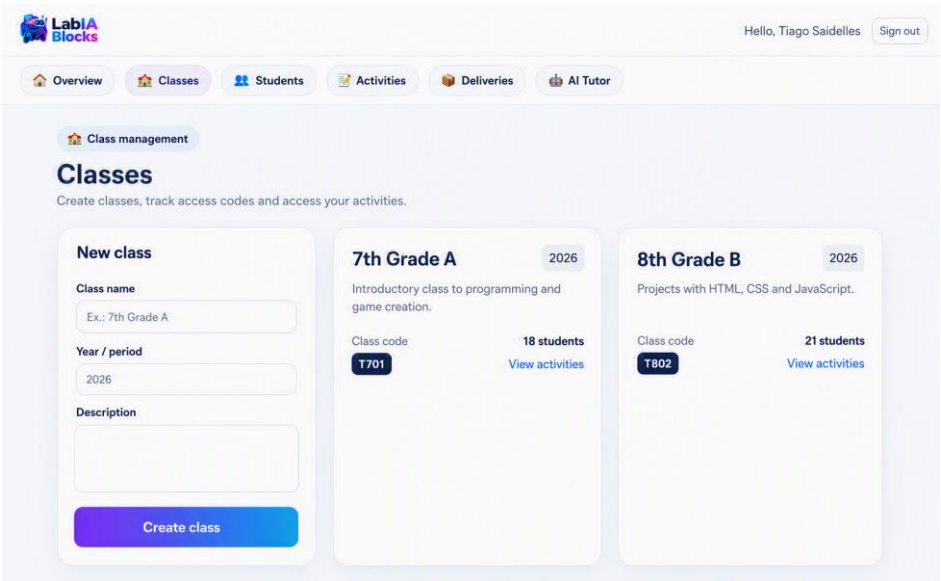
Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

The use of complete examples is particularly relevant for students who are not yet able to plan a game from scratch. Rather than treating the model as the final answer, teachers can turn it into an object of investigation by asking students to identify which blocks create the interface, where the score is updated, how collisions are detected, which conditions determine victory and defeat, or how a rule can be modified. The logic of creation and remixing is consistent with a perspective on digital fluency that emphasizes producing and transforming artifacts rather than merely consuming technology (Resnick et al., 2009, pp. 60–67).

FEATURES OF THE TEACHER AREA

The teacher area organizes the use of the tool as a classroom environment. Teachers can create classes with access codes, register and manage students, publish activities, and monitor submissions. When creating an activity, teachers provide the class, title, instructions, and deadline. They can also select the level of AI assistance from five options: 0 – No AI; 1 – Questions; 2 – Hints; 3 – Tutor; and 4 – Laboratory. This scale makes it possible to adapt the support to the objective of the activity by restricting assistance when greater autonomy is desired or expanding it in situations involving guided exploration. Student submissions are linked to projects and activities. Teachers can assign grades and record feedback comments, changing the status of a submission to reviewed. There is also a specific area for configuring the OpenAI key, selecting the model, and enabling the AI Tutor globally. The architecture keeps the key out of the student's browser, which is an important technical quality for institutional use, although schools must still adopt their own data security, privacy, and governance policies. From a pedagogical perspective, teacher management can be understood as an integrated three-stage process: creating a class, registering students, and then publishing the activity that will guide the learning experience. This organization allows teachers to transform a programming proposal into a monitorable sequence by linking each student to a group, each challenge to a class, and each submission to a specific activity. Rather than using the editor merely as an isolated resource, the platform begins to function as an environment for lesson planning, development, and monitoring.

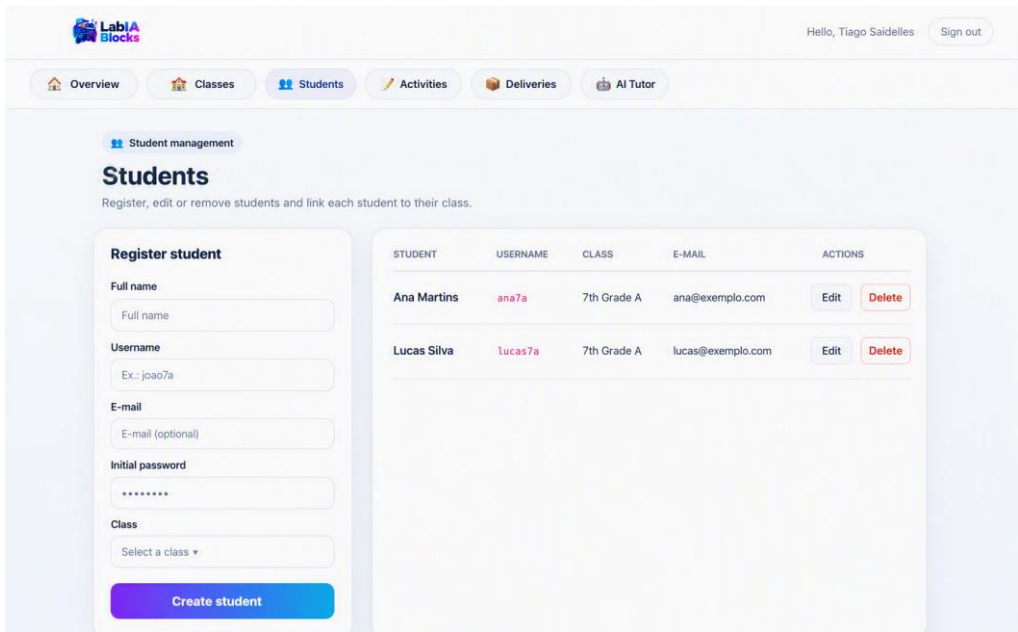
Figure 7
Class registration and organization in the teacher dashboard



Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

The class-creation process shown in Figure 7 establishes the organizational unit for the work. The teacher enters a name, academic period, and description, and the system generates a code associated with the class. In a pedagogical application, this feature can be used to distinguish groups by grade, workshop, project, or level of complexity. The class also serves as a reference for assigning activities and monitoring student work, allowing the same tool to be used in different contexts without mixing student records.

Figure 8
Student registration, assignment, and management

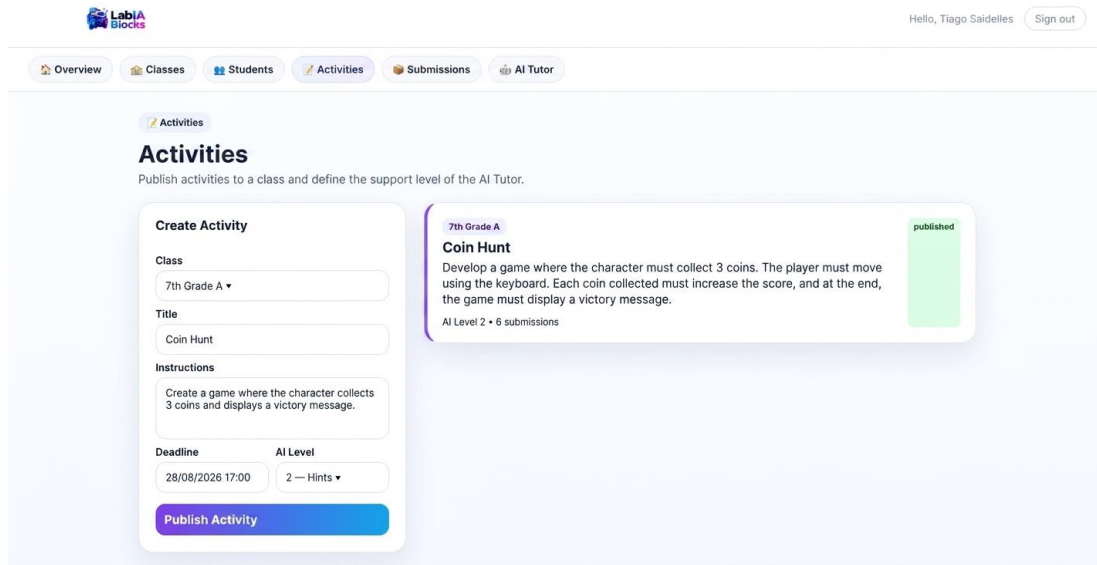


Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

In the next stage, teachers register students, define their usernames and initial passwords, and select the class to which each student will be assigned, as shown in Figure 8. The ability to edit or delete records facilitates class management throughout the academic period. From a pedagogical standpoint, individual registration enables monitored learning pathways: each student accesses a personal space, receives the relevant activity, and develops projects within a structure that enables teachers to observe submissions and provide feedback. User management thus ceases to be merely an administrative function and begins to support personalization and formative monitoring.

Figure 9

Publishing an activity with a class, instructions, deadline, and AI support level



The screenshot shows the 'LabIA Blocks' web interface. At the top, there's a navigation bar with 'Overview', 'Classes', 'Students', 'Activities' (selected), 'Submissions', and 'AI Tutor'. A user profile 'Hello, Tiago Saidelles' with a 'Sign out' button is on the right. The main content area is titled 'Activities' with the subtitle 'Publish activities to a class and define the support level of the AI Tutor.' On the left is a 'Create Activity' form with fields for 'Class' (7th Grade A), 'Title' (Coin Hunt), 'Instructions' (Create a game where the character collects 3 coins and displays a victory message.), 'Deadline' (28/08/2026 17:00), and 'AI Level' (2 — Hints). A 'Publish Activity' button is at the bottom. On the right is a 'published' activity card for '7th Grade A' titled 'Coin Hunt' with the description 'Develop a game where the character must collect 3 coins. The player must move using the keyboard. Each coin collected must increase the score, and at the end, the game must display a victory message.' and 'AI Level 2 • 6 submissions'.

Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

Publishing activities completes this process. As shown in Figure 9, teachers select the class, define a title, describe the mission, establish a deadline, and choose the AI Tutor support level. This configuration allows mediation to be adapted to the pedagogical purpose. In an introductory activity, for example, a higher level of guidance may be offered to help students recognize the function of the blocks. In subsequent activities, support may be reduced to increase decision-making and authorship. The activity statement may also specify success criteria, such as moving a character, using a variable, constructing a victory condition, and testing the game before submission. Creating the activity therefore establishes the learning objectives, completion criteria, and degree of support. One possible classroom strategy is for the teacher to prepare the class and access credentials in advance, publish a short mission, and begin the lesson by reading the problem collectively. Students can then record the logic in Portugol, construct the structure in HTML, work on its presentation in CSS, and complete the interaction in JavaScript. During development, the teacher monitors recurring difficulties and decides when to intervene directly or advise students to consult the AI Tutor. At the end, the submission may be accompanied by an explanation of the solution adopted, making it possible to assess not only whether the game works but also whether students

understand the relationships among the blocks used, the configured values, and the behavior produced in the game.

Table 2

Examples of Potential Pedagogical Activities Using the Platform

Type of Activity	Teacher Action	Student Action	Concepts Involved
Guided assembly	Provide an objective and limit the level of AI assistance.	Assemble the specified sequence and test the result.	HTML structure, CSS styling, events, and variables.
Debugging challenge	Provide a project containing an error or incomplete rule.	Use execution, code, and the AI Tutor to identify the error.	Debugging, conditions, state, and causality.
Game project	Define a theme, criteria, and deadline.	Plan, create, test, revise, and submit a game.	Decomposition, logic, authorship, and design.
Portugol to blocks	Require an algorithm before implementation.	Write the logic and convert it into blocks.	Algorithms, sequence, conditions, and repetition.
Remixing	Select a ready-made game and propose modifications.	Examine the existing structure and modify its mechanics.	Code reading, reuse, and creativity.
Gamified challenges	Define a sequence of challenges and criteria.	Meet progressive goals and monitor feedback.	Progression, autonomy, and metacognition.
Student-created quiz	Request a question-based game about curricular content.	Program questions, answer choices, scoring, and lives.	Events, conditionals, interface, and formative assessment.
Project presentation	Ask students to explain how the game works.	Display the blocks and code and explain design decisions.	Communication, justification, and computational thinking.

Source: Prepared by the authors (2026).

POTENTIAL PEDAGOGICAL QUALITIES AND BENEFITS

The first quality identified is the coherence between visual representation and textual language.

The environment does not present block-based programming as a domain separate from HTML, CSS, and JavaScript. Instead, it organizes the blocks according to these languages and allows students to view the resulting code. This characteristic may reduce the disruption commonly experienced when students transition from an exclusively visual environment to a written programming language. The literature on block-based environments suggests that visual representations can support initial learning by reducing syntax errors and facilitating the recognition of structures, provided that the transition and conceptual understanding are appropriately planned (Weintrop; Wilensky, 2018, pp. 1–25). The second quality is product-oriented learning. Students do not assemble abstract sequences merely to obtain numerical outputs; they construct an interactive experience with a setting, characters, rules, and objectives. The visible result gives immediate meaning to their changes. Such a dynamic may promote motivation and

persistence, particularly when teachers organize the work as a project involving intermediate goals, revision, and a final presentation. This interpretation is consistent with active learning approaches in which students participate in constructing solutions and receive feedback throughout the process (Freeman et al., 2014, pp. 8410–8415).

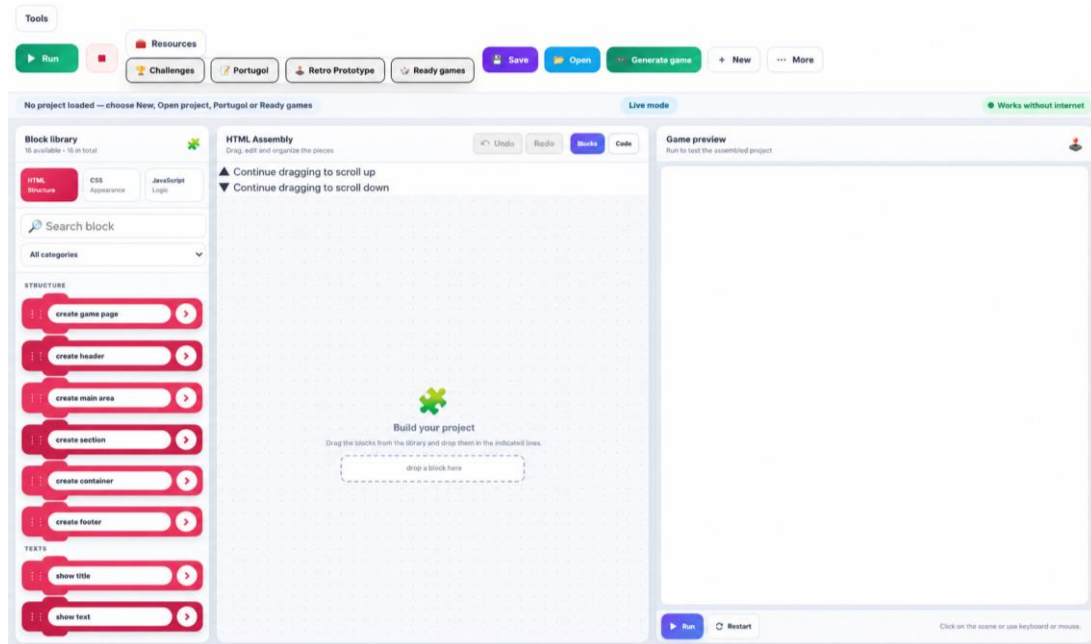
The third quality is the possibility of pedagogical differentiation. The same environment can be used in free mode, through challenges, with ready-made models, through planning in Portugal, or within an activity defined by the teacher. In addition, the level of AI assistance can vary among activities. In a heterogeneous class, teachers can propose the same objective with different degrees of support: some students may work from a ready-made example, others may receive only hints, and others may be challenged to create the solution from the outset. The fourth quality concerns feedback. Students have access to immediate game execution, challenge messages, analysis from the AI Tutor, and the ability to compare blocks and code. Teachers, in turn, can monitor submissions and record assessments and comments. This combination creates different layers of feedback originating from the system itself, the AI, and the teacher. The educational potential of AI nevertheless depends on supervision and the design of tasks that require explanation, revision, and justification, thereby preventing automated responses from replacing understanding. This caution is consistent with the risks highlighted by Humble (2024) and UNESCO's (2024) recommendations for human-centered and pedagogically validated use. The fifth quality is the presence of gamification elements integrated into the learning process. Unlockable challenges, progress, achievements, and ready-made games make the experience resemble a mission-based pathway. The pedagogical use of these elements may promote engagement when progress represents clear learning objectives. The tool thus enables teachers to employ gamification without reducing the activity to points and badges because the verified goals are associated with concrete programming actions.

TEACHING STRATEGY FOR USING LABIA BLOCKS IN THE CLASSROOM

The pedagogical use of LabIA Blocks can be organized as a progressive construction pathway in which students move from understanding the problem to representing it algorithmically and subsequently implementing the game. The recommended starting point is not the random selection of blocks but the definition of a clear mission: what the player must do, which objects exist, which rules determine success or failure, and which actions must occur in response to each event. This initial stage promotes problem decomposition and brings the activity closer to computational thinking practices in which problems are organized into smaller parts, relationships, and procedures before implementation (Grover; Pea, 2013, pp. 38–43). When preparing the lesson, teachers can organize the class, register students, and publish an activity linked to that group in advance. The activity can present the challenge, expected product, deadline, submission criteria, and AI Tutor support level. This stage is important because it makes the pedagogical purpose explicit before students open the editor. Instead of beginning by randomly selecting blocks, students receive a mission with defined objectives and can plan how to represent the solution in HTML, CSS, and JavaScript. Teachers, in turn, can adjust task complexity and monitor the development of submissions throughout an instructional sequence. A possible instructional sequence for students begins with reading the mission and planning in Portugol. At this stage, the aim is to record actions in language resembling ordinary speech, such as starting the game, creating a character, moving when a key is pressed, checking for collisions, changing the score, and testing the victory condition. Students then access the editor and begin with the HTML structure, selecting blocks that organize the page, setting, titles, text, scoreboard, and other necessary elements. The next step focuses on CSS, when appearance, spatial organization, dimensions, and visual identity are addressed. Only after establishing this foundation do students implement events, variables, conditions, repetition, collisions, and game rules in JavaScript.

Figure 10

Editor in its initial state: block library, assembly area, and game preview



Source: Prepared by the authors based on the analyzed version of LabIA Blocks (2026). Interface text translated and image adapted.

Figure 10 highlights an important characteristic for lesson organization: the block library, assembly area, and preview space remain available simultaneously. This arrangement promotes a short action-and-feedback cycle. Students select a block, configure its values, connect it to the sequence, run the project, and observe the consequence. If the behavior differs from what was expected, they can return to the assembly and revise the structure. In this context, errors can be addressed as part of the learning process rather than merely as negative outcomes. Teachers can explore this cycle by asking students to formulate hypotheses before each change and subsequently explain why a particular modification produced the observed result. The AI Tutor can be incorporated primarily during moments of uncertainty, debugging, and revision. A pedagogically productive use consists of asking for explanations about the function of a particular block, requesting hints to identify an error, or comparing two ways of implementing a mechanic. The response received must be tested in the editor itself and compared with the game's behavior. In this way, AI becomes part of an investigative process rather than a substitute for execution. UNESCO (2024) recommends that the use of generative AI in education be guided by

pedagogical purpose, human supervision, and the development of critical abilities. In LabIA Blocks, these conditions can be operationalized through teacher control over assistance levels and the requirement that suggestions be validated in practice.

At the end of an activity, students should present not only a functioning game but also the logic employed. They can explain which blocks structure the interface, which events receive keyboard commands, which variables store states, how conditions control victory or defeat, and which changes were made during debugging. This presentation transforms the digital product into evidence of the learning process and enables teachers to assess understanding, authorship, and the ability to justify decisions. The practice also helps address a recurring difficulty in tasks involving artificial intelligence: distinguishing a solution that was merely copied from one that was understood and modified by the student.

PROPOSED SEQUENCE OF ACTIVITIES AND FORMATIVE ASSESSMENT

LabIA Blocks makes it possible to structure a sequence of activities that begins with short tasks and advances toward original projects. In an initial stage, teachers can propose interface-recognition activities and guided exploration of HTML, CSS, and JavaScript blocks. The objective is not to build a complex game immediately but to understand the function of each group of commands. A simple activity may ask students to create a title, a message, and a setting; another may require them to change colors and dimensions; and a third may introduce keyboard movement. Each task adds a layer of complexity and provides a repertoire for the subsequent project. In a second stage, logic challenges can address events, variables, and conditions separately. Students can create a counter, make an object disappear after a collision, limit the number of lives, or end the game when a variable reaches a specific value. Progression through small goals tends to make causal relationships visible before they are combined into larger mechanics. This organization is consistent with active methodologies because students engage with concrete problems, test solutions, and receive feedback throughout the process. These characteristics have

been associated with better outcomes in active learning contexts compared with exclusively passive exposition (Freeman et al., 2014, pp. 8410–8415). Figure 9 illustrates how teachers can transform a programming competency into a learning mission. Rather than simply asking students to “make a game,” the activity can specify mechanics, functional criteria, and mandatory elements. A collection challenge, for example, may require keyboard movement, three independent items, score updates, and a victory message after the final item is collected. An objective description facilitates assessment because the same requirements used to guide construction can form part of an assessment rubric. The AI assistance level, defined when the activity is published, becomes part of the instructional plan and may vary according to complexity and the intended degree of autonomy.

Table 3

Suggested Instructional Sequence for Introductory Programming and Game Creation

Stage	Instructional Objective	Student Action	Assessment Evidence
1. Plan	Decompose the game into entities, rules, and objectives.	Describe the mission and represent the logic in Portugol.	A coherent sequence of actions and conditions.
2. Structure with HTML	Understand how interface elements are organized.	Assemble the setting, text, scoreboard, and necessary components.	A structure compatible with the proposed game.
3. Style with CSS	Relate visual properties to the gaming experience.	Configure colors, dimensions, arrangement, and appearance.	A legible interface consistent with the proposal.
4. Program with JavaScript	Implement behavior, events, and states.	Create movement, variables, collisions, and rules.	Mechanics executed according to the requirements.
5. Test and debug	Develop error-analysis and revision skills.	Run the project, identify errors, request hints, and correct them.	A record or explanation of the changes made.
6. Present and remix	Consolidate authorship and explanatory skills.	Demonstrate the game, justify the blocks used, and propose improvements.	Presentation, functionality, and reflection on the project.

Source: Prepared by the authors (2026).

Table 3 can be used across a sequence of sessions or adapted to an intensive workshop. In introductory classes, the three languages can be introduced progressively, preventing the large number of available blocks from becoming a source of overload. In more experienced classes, teachers can propose a single project and use the stages as monitoring milestones. This flexibility is particularly useful for group activities: participants can discuss game rules, test different configurations, and compare solutions, but

assessment should preserve individual evidence of understanding, such as an oral explanation, a brief written justification, or the modification of a rule in the teacher's presence.

Formative assessment can combine the game's functionality with the development process. At least five dimensions should be considered: understanding of the problem, selection and organization of blocks, coherence of the logic, ability to test and debug, and explanation of decisions. Aesthetic elements may be valued, but they should not obscure the central objective of understanding programming structures. Similarly, use of the AI Tutor can be considered part of the process when students demonstrate that they analyzed the suggestion, executed the solution, and made their own adjustments. The emphasis therefore falls on supported authorship rather than merely on the presence or absence of artificial intelligence. Gamification elements can be integrated into this assessment without turning learning exclusively into a competition for points. Challenges, achievements, progression, and missions can indicate completed stages, provide short-term objectives, and recognize progress. Deterding et al. (2011, pp. 9–15) define gamification as the use of game design elements in nongame contexts. In the educational context analyzed here, its value depends on progression representing actual learning. Unlocking a challenge may therefore mean that a student has demonstrated a competency, rather than merely having remained connected to the platform for a longer period.

LIMITATIONS AND CONDITIONS FOR EDUCATIONAL USE

Although the analysis reveals a broad set of resources, the tool should not be interpreted as a substitute for teachers or as an automatic guarantee of learning. The first limitation arises from the nature of the present study: no data were collected on student performance, engagement, or learning. Based solely on the functional analysis, it is therefore not possible to claim that LabIA Blocks increases grades, reduces dropout rates, or produces measurable gains in computational thinking. Such hypotheses must be investigated through implementation studies. The second limitation concerns the AI Tutor. Even with validation based on the block catalog and semantic rules, generative models may produce inappropriate

responses, interpret ambiguities differently from the student's intention, or suggest a solution that is technically possible but pedagogically undesirable. Teachers must maintain monitoring criteria and encourage students to execute, test, and justify the responses they receive. The tool makes a greater contribution when AI is treated as a source of hints and explanations subject to verification rather than as a final authority.

The third condition concerns infrastructure. The editor was designed to operate locally, but the portal uses PHP/MySQL, and Bootstrap components may depend on a content delivery network, depending on the installation. The AI Tutor also requires connectivity and a configured API when enabled. In schools with limited connectivity, an entirely local deployment must include the necessary front-end resources and provide for activities that remain functional without the Tutor. The ability to operate the editor and save projects in JSON provides an important alternative in such contexts.

CONCLUSION

This study described the construction and features of LabIA Blocks and analyzed its potential for introductory education in HTML, CSS, JavaScript, and game development. The platform combines visual block-based programming, code visualization, immediate execution, planning in Portugol, progressive challenges, ready-made games, achievements, activity management, and an AI Tutor within a single environment. Analysis of the catalog identified 108 blocks, ranging from basic structural and styling elements to events, variables, lists, collisions, scoring, lives, and game mechanics. For teachers, the tool provides class and student organization, activity creation with deadlines and differentiated levels of AI support, submission monitoring, and the recording of assessments and feedback. For students, it offers a pathway that may begin with examples and challenges and progress toward original projects, allowing them to observe the relationship between blocks and textual code. This combination creates conditions for project-based learning, problem-solving, debugging, and remixing activities while employing gamification elements such as progression, challenges, and achievements. The main potential pedagogical

contribution of LabIA Blocks lies in its integration of different forms of representation and support: planning in algorithmic language, assembling visually, reading code, running the game, receiving feedback, and revising the solution. The AI Tutor can expand this mediation when used contextually, with assistance levels defined by the teacher and the preservation of student authorship. Nevertheless, AI does not eliminate the need for verification, teacher mediation, and the development of critical thinking.

As a continuation of this research, implementation studies should be conducted with basic education classes and introductory programming education, comparing different strategies for using the tool. Future research may investigate the learning of specific concepts, project quality, debugging patterns, teacher and student perceptions, levels of autonomy, and the influence of the different AI Tutor assistance levels. This stage will permit a transition from the analysis of pedagogical potential to empirical evidence concerning the conditions, limitations, and outcomes associated with the use of LabIA Blocks in real educational contexts.

REFERENCES

- Deterding, Sebastian; Dixon, Dan; Khaled, Rilla; Nacke, Lennart. From game design elements to gamefulness: defining gamification. In: *PROCEEDINGS OF THE 15TH INTERNATIONAL ACADEMIC MINDTREK CONFERENCE*. Tampere: ACM, 2011. p. 9-15. DOI: <https://doi.org/10.1145/2181037.2181040>. Accessed on: 13 Aug. 2026.
- Freeman, Scott; Eddy, Sarah L.; McDonough, Miles; Smith, Michelle K.; Okoroafor, Nnadozie; Jordt, Hannah; Wenderoth, Mary Pat. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, v. 111, n. 23, p. 8410-8415, 2014. DOI: <https://doi.org/10.1073/pnas.1319030111>. Accessed on: 13 Aug. 2026.
- Grover, Shuchi; Pea, Roy. Computational thinking in K-12: a review of the state of the field. *Educational Researcher*, v. 42, n. 1, p. 38-43, 2013. DOI: <https://doi.org/10.3102/0013189X12463051>. Accessed on: 13 Aug. 2026.

- Humble, Niklas. Risk management strategy for generative AI in computing education: how to handle the strengths, weaknesses, opportunities, and threats? *International Journal of Educational Technology in Higher Education*, v. 21, art. 61, 2024. DOI: <https://doi.org/10.1186/s41239-024-00494-x>. Accessed on: 13 Aug. 2026.
- Resnick, Mitchel; Maloney, John; Monroy-Hernández, Andrés; Rusk, Natalie; Eastmond, Evelyn; Brennan, Karen; Millner, Amon; Rosenbaum, Eric; Silver, Jay; Silverman, Brian; Kafai, Yasmin. Scratch: programming for all. *Communications of the ACM*, v. 52, n. 11, p. 60-67, 2009. DOI: <https://doi.org/10.1145/1592761.1592779>. Accessed on: 13 Aug. 2026.
- UNESCO. *Guia para a IA generativa na educação e na pesquisa* [Guidance for generative AI in education and research]. Paris: UNESCO, 2024. ISBN 978-92-3-700028-1. Available at: <https://unesdoc.unesco.org/ark:/48223/pf0000390241>. Accessed on: 13 Aug. 2026.
- Weintrop, David; Wilensky, Uri. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education*, v. 18, n. 1, art. 3, p. 1-25, 2018. DOI: <https://doi.org/10.1145/3089799>. Accessed on: 13 Aug. 2026.